Computations under Time Constraints: Algorithms Developed for Fuzzy Computations Can Help

Karen Villaverde
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
Email: kvillave@cs.nmsu.edu

Olga Kosheleva and Martine Ceberio
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
Emails: {olgak,mceberio}@utep.edu

***Abstract*—In usual computers – that use binary representation of real numbers – an irrational real number (and even a rational number like 1.3 or 1.2) can only be computed with a finite accuracy. The more accuracy we need, the larger the computation time. It is therefore reasonable to characterize the complexity of computing a real number a by the accuracy $\Delta_a(t)$ that we can achieve in time t. Once we know this characteristic for two numbers a and b, how can we compute a similar characteristic for, e.g., $c = a + b$? In this paper, we show that the problem of computing this characteristic can be reduced to the problem of computing the membership function for the sum – when we use Zadeh's extension principle with algebraic product as the "and"-operation. Thus, known algorithms for computing this membership function can be used to describe computations under time constraints.**

I. Formulation of the Problem

Computing real numbers: a problem. In many practical applications, a physical theory described the value of the desired quantity in precise mathematical terms. For example, it is known:

- that the length ℓ of the diagonal of a unit square is equal to $\ell = \sqrt{2}$;
- that the circumference c of a circle of unit radius is

$$c = 2\pi;$$

- that for a normally distributed random variable with 0 mean and standard deviation σ, the probability P that this variable exceed 3σ is equal to

$$P = \frac{1}{2\pi} \cdot \int_3^\infty \exp\left(-\frac{x^2}{2}\right)\, dx.$$

Often, we need to perform additional processing that uses these values: for example, we can use the probability to compute the expected value of the losses and profits and thus, make a decision about a investment. To be able to perform such processing, we need to represent the original values in the computer. Often, we need to perform computations to produce such a representation. Computations needed to produce this representation are called *computing* the desired real number.

All the real numbers represented in the existing computers are rational numbers, specifically, *binary rational* numbers of the form $\frac{p}{2^q}$, where p and q are integers. Thus, when a number that we want to compute is irrational (e.g., $\sqrt{2}$ or π) or rational but not binary rational (e.g., 1/3 or 1.2), we can only represent a rational approximation to this number. In other words, an irrational real number can only be represented (and, thus, computed) with a finite accuracy.

In precise terms, the problem is as follows. We are given a mathematical description of a real number x – such as either $\sqrt{2}$, or the above integral, or value $x(t_0)$ where t_0 is a given number and $x(t)$ is a solution to a given differential equation with given initial conditions. The problem is:

- given a rational number ε,
- compute a binary-rational number r for which $|x - r| \le \varepsilon$.

We can then say that r represents x with accuracy ε, or that, by producing r, we have computed x with accuracy ε.

In each of the above examples, there exists an algorithm that, given a rational number $\varepsilon > 0$, computes x with accuracy ε (i.e., an algorithm that, given $\varepsilon > 0$, produces a binary-rational number r for which $|x - r| \le \varepsilon$). Real numbers for which such an algorithm is possible are called *computable*.

Need to take time constraints into account. The more accuracy we need, the larger the computation time. It is therefore reasonable to characterize the complexity of computing a real number a by the accuracy $\Delta_a(t)$ (e.g., 10^{-3}, 10^{-8}, etc.) that we can achieve in time t.

We are interested in algorithmically computable numbers, i.e., numbers that we can, in principle, compute with an arbitrary accuracy. For such numbers, the accuracy $\Delta_a(t)$ tends to 0 as $t \to \infty$.

How to find accuracy of the result of data processing? A general problem. As we have mentioned earlier, the main reason why we compute different real numbers a, b, ..., in the first place is that later on, we may be interested in computing numbers of the type $c = f(a, b, \ldots)$ for some computable function $f(a, b, \ldots)$ from real numbers (or tuples of real numbers) into real numbers. Computing c based on the known results of computing a, b, ..., is usually called *data processing*.

In other words, data processing is the last stage in computing the real number c. For example, if we have previously computed 2π, then, when we compute the above integral, we can use the result of computing 2π.

Thus, once we know the characteristics $\Delta_a(t)$, $\Delta_b(t)$, ..., that describe the complexity of computing the values a, b, ..., it is desirable to compute a similar characteristic $\Delta_c(t)$ for the new number $c = f(a, b, \ldots)$.

This is the problem that we analyze in this paper.

The simplest case: estimating time complexity of the sum of two numbers. Before we discuss the general problem, we will pay a special attention to the simplest case of the above problem. The simplest case is when we have the simplest function $f(a, b) = a+b$. In this case, we arrive at the following problem:

- we know the characteristics $\Delta_a(t)$ and $\Delta_b(t)$ corresponding to numbers a and b, and
- we want to find the characteristic $\Delta_c(t)$ corresponding to $c = a + b$.

Important comment. In this paper, we consider situations in which computing $c = a+b$ consists of the following two steps:

- first, we computing a and b (with some accuracy), and
- then, we add the resulting approximations $\widetilde{a}$ and $\widetilde{b}$ to a and b.

It is worth mentioning that this is not always the fastest way to compute the number c. For example, when $a = \pi$ and $b = -\pi$, then $c = a+b = 0$; in this case, a straightforward computation of $c = 0$ is much faster than computing $a = \pi$ first.

II. Estimating Time Complexity of the Sum of Two Numbers: Formulas

Analysis of the problem. According to the above assumption, if we want to compute $c = a + b$ in time t, we should spend some time computing a, then some time computing b, and then some time adding a and b.

Let us denote the time that we spend on computing a by t_a and the time that we spend on computing b by t_b. For simplicity, we assume that computing the sum requires a single computation step and that we measure time not in seconds, but in such steps. Under these assumptions, adding two approximations requires 1 unit of time. The total computation time t is equal to the sum of the computation times required for all three steps, i.e., $t = t_a + t_b + 1$.

By definition of the characteristic $\Delta_a(t)$, during the time t_a, at best we can compute a with an accuracy $\Delta_a(t_a)$. In other words, at best, we can compute an approximation $\widetilde{a}$ for which $|\widetilde{a} - a| \le \Delta_a(t_a)$. Similarly, during the time t_b, at best we can compute b with an accuracy $\Delta_b(t_b)$. In other words, at best, we can compute an approximation $\widetilde{b}$ for which $|\widetilde{b} - b| \le \Delta_b(t_b)$. From these inequalities, we conclude that the difference between the computed value $\widetilde{c} = \widetilde{a} + \widetilde{b}$ and the desired value $c = a + b$ is bounded by

$$|\widetilde{c} - c| = |(\widetilde{a} + \widetilde{b}) - (a + b)| = |(\widetilde{a} - a) + (\widetilde{b} - b)| \le$$

$$|\widetilde{a} - a| + |\widetilde{b} - b| \le \Delta_a(t_a) + \Delta_b(t_b).$$

Thus, the most accurate approximation occurs when we select t_a and t_b so as to guarantee the smallest possible bound $\Delta_a(t_a) + \Delta_b(t_b)$. In other words, we arrive at the following expression.

Resulting formula:

$$\Delta_c(t) = \min_{t_a, t_b:\ t_a + t_b + 1 = t} (\Delta_a(t_a) + \Delta_b(t_b)). \tag{1}$$

Important simplification. Computation is a problem when $t_a \gg 1$ and $t_b \gg 1$. In this case, we can safely ignore "1" in the formula $t_a + t_b + 1 = t$ and replace it with an approximate equality $t_a + t_b = t$. In this approximation, the above formula (1) takes the following simplified form:

$$\Delta_c(t) = \min_{t_a, t_b:\ t_a + t_b = t} (\Delta_a(t_a) + \Delta_b(t_b)). \tag{2}$$

What we plan to do. In this paper, we show that the existing algorithms for processing fuzzy numbers can be used to find the characteristic (2).

III. Processing Fuzzy Numbers: Brief Reminder

Need to process fuzzy numbers. In many practical situations, we have some information about the quantities $x_1, \ldots, x_n$, and we are interested in a quantity y that is related to x_i by a known dependence $y = f(x_1, \ldots, x_n)$. It is therefore necessary to find out what information about y we can deduce from the known information about x_i.

An important particular case of this general problem is when we have fuzzy information about x_i; see, e.g., [3], [7]. In this case, for each quantity x_i, instead of an exact value x_i, we have a fuzzy number X_i characterizing this property, i.e., for every value x_i, we know the degree $\mu_i(x_i)$ to which this value is possible, i.e., to which it is possible that $X_i = x_i$.

Using a standard notation $\diamondsuit$ for "possible" from modal logic, we can describe the statement $S \stackrel{\text{def}}{=}$"it is possible that $X_i = x_i$" as $\diamondsuit(X_i = x_i)$. In these terms, the value $\mu_i(x_i)$ is our degree of confidence $d(S)$ in this statement S: $\mu_i(x_i) = d(\diamondsuit(X_i = x_i))$.

Zadeh's extension principle: derivation. For $Y = f(X_1, \ldots, X_n)$, the value y is possible if and only if there exist values $x_1, \ldots, x_n$ for which it is possible that $X_1 = x_1$, ..., it is possible that $X_n = x_n$, and $y = f(x_1, \ldots, x_n)$. In other words,

$$\diamondsuit(Y = y) \Leftrightarrow \exists x_1, \ldots, x_n\, (y = f(x_1, \ldots, x_n)\,\&$$

$$\diamondsuit(X_1 = x_1)\,\&\ \ldots\ \&\ \diamondsuit(X_n = x_n)).$$

We want to estimate our degree of confidence $\mu(y) = d(\diamondsuit(Y = y))$ in this statement.

We know the degrees of confidence

$$\mu_i(x_i) = d(\diamondsuit(X_i = x_i))$$

in individual statements $\diamondsuit(X_i = x_i)$. Following the general ideas of fuzzy logic, we can then use a t-norm (a fuzzy "and")

operation $f_{\&}(a,b)$ to describe the degree of confidence in the conjunction

$$\diamondsuit(X_1 = x_1)\,\&\,\ldots\,\&\,\diamondsuit(X_n = x_n):$$

namely,

$$d(\diamondsuit(X_1 = x_1)\,\&\,\ldots\,\&\,\diamondsuit(X_n = x_n)) =$$
$$f_{\&}(d(\diamondsuit(X_1 = x_1)),\ldots,d(\diamondsuit(X_n = x_n)) =$$
$$f_{\&}(\mu_1(x_1),\ldots,\mu_n(x_n)).$$

The existential quantifier is, in effect, an infinite "or" statement: namely, the above statement means that either the formula $y = f(x_1,\ldots,x_n)\,\&\,\diamondsuit(X_1 = x_1)\,\&\,\ldots$ is true for one tuple, or from another tuple, etc. So, to combine the degrees $f_{\&}(\mu_1(x_1),\ldots,\mu_n(x_n))$ into the desired degree $\mu(y)$, we must use a t-conorm (fuzzy "or" operation) $f_\vee(a,b)$.

For most t-conorms, we have $f_\vee(a,a) < a$, so when we apply it infinitely many times, the resulting degree tends to 0. The only case when we get a non-zero result is when we use the maximum t-norm $f_\vee(a,b) = \max(a,b)$. In this case, the desired degree $\mu(y) = d(\diamondsuit(Y = y))$ is equal to the largest of the values $f_{\&}(\mu_1(x_1),\ldots,\mu_n(x_n))$ for all the tuples $(x_1,\ldots,x_n)$ for which $f(x_1,\ldots,x_n) = y$. Thus, we arrive at the following formula:

Zadeh's extension principle: resulting general formula. Once we know the membership functions $\mu_1(x_1),\ldots,\mu_n(x_n)$ corresponding to n variables $x_1,\ldots,x_n$, the membership function $\mu(y)$ corresponding to $y = f(x_1,\ldots,x_n)$ takes the form

$$\mu(y) = \max_{x_1,\ldots,x_n:\ f(x_1,\ldots,x_n)=y} f_{\&}(\mu_1(x_1),\ldots,\mu_n(x_n)). \quad (3)$$

The most most widely used fuzzy "and"-operations are the minimum $f_{\&}(a,b) = \min(a,b)$ and the algebraic product $f_{\&}(a,b) = a\cdot b$. Thus, we arrive at the following formulas:

$$\mu(y) = \max_{x_1,\ldots,x_n:\ f(x_1,\ldots,x_n)=y} \min(\mu_1(x_1),\ldots,\mu_n(x_n)), \quad (4)$$

which is the most widely used form of Zadeh's extension principle, and

$$\mu(y) = \max_{x_1,\ldots,x_n:\ f(x_1,\ldots,x_n)=y} \mu_1(x_1)\cdot\ldots\cdot\mu_n(x_n). \quad (5)$$

Simplest case of addition. For the simplest case of the addition function $f(t_a,t_b) = t_a + t_b$, the above formulas take the form

$$\mu(t) = \max_{t_a,t_b:\ t_a+t_b=t} f_{\&}(\mu_a(t_a),\mu_b(t_b)); \quad (6)$$

$$\mu(t) = \max_{t_a,t_b:\ t_a+t_b=t} \min(\mu_a(t_a),\mu_b(t_b)); \quad (7)$$

and

$$\mu(t) = \max_{t_a,t_b:\ t_a+t_b=t} \mu_a(t_a)\cdot\mu_b(t_b). \quad (8)$$

Straightforward computation of the expression (8). In reality, we can only know the values of $\mu_a(x)$ and $\mu_b(x)$ for finitely many values x. Let us denote the total number of such values by n. In this case, it is reasonable to compute only n values of $\mu(t)$. For each of these n values, according to the formula (8), we must find the largest of n products. Computing each product takes 1 elementary computational step, computing the largest of n numbers requires that we do $n-1$ comparisons. So, the total number of computation steps that needs to be done to compute one value of $\mu(t)$ is $2n-1 = O(n)$. Thus, to compute *all* n values of the desired membership function $\mu(t)$, we need $n\cdot O(n) = O(n^2)$ computational steps.

For large n, this number if large, so it is desirable to have faster algorithms for computing this expression.

A faster algorithm for computing the expression (8): main idea. Such faster algorithms are known. For example, an algorithm described in [5], [6] is based on the well-known fact that for non-negative numbers $\mu_1,\ldots,\mu_n$, we have

$$\max(\mu_1,\ldots,\mu_n) = \lim_{p\to\infty} (|\mu_1|^p + \ldots + |\mu_n|^p)^{1/p}$$

(see, e.g., [4]). Therefore, for sufficiently large p, we have

$$\max(\mu_1,\ldots,\mu_n) \approx (|\mu_1|^p + \ldots + |\mu_n|^p)^{1/p};$$

the larger p, the better the quality of this approximation.

Applying this approximate formula to the values $\mu_a(t_a)\cdot\mu_b(t-t_a)$ maximized in the formula (8), we come up with an approximate formula $\mu(t) \approx M(t)^{1/p}$, where we denoted

$$M(t) = \sum_{t_a} (\mu_a(t_a)\cdot\mu_b(t-t_a))^p.$$

The formula for $T(t)$ can be rewritten as:

$$M(t) = \sum_{t_a} (\mu_a(t_a))^p\cdot(\mu_b(t-t_a))^p.$$

In the natural assumption that the values t_a are equally spaced, with step h, this sum becomes a *convolution* of two functions: $M_a(x) = (\mu_a(x))^p$ and $M_b(x) = (\mu_b(x))^p$. Now, we can use the following two ideas to compute $M(x)$ fast:

- It is known that the Fourier transform of the convolution $M_a * M_b$ of two functions M_a and M_b is equal to the product of their Fourier transforms.
- Fourier transform can be computed in time $O(n\log(n))$ [8], [9]; the corresponding algorithms are called *Fast Fourier Transform* (FFT, for short).

In view of these two facts, we can use the following algorithm to compute the membership function that expresses the sum of two given fuzzy numbers:

Given: the values $\mu_a(t_a)$ and $\mu_b(t_b)$ for n equally spaced values t_a and t_b.

Algorithm: First, we pick a large number p (the larger p, the better the results of our computations). Then, we do the following:

1) For each of n values t_a, we compute the values $M_a(x) = (\mu_a(x))^p$ and $M_b(x) = (\mu_b(x))^p$.

2) We apply FFT to the functions $M_a(x)$ and $M_b(x)$ and get their Fourier transforms $\hat{M}_a(\omega)$ and $\hat{M}_b(\omega)$ (for n different values ω).
3) We multiply $\hat{M}_a(\omega)$ and $\hat{M}_b(\omega)$; let us denote the corresponding product by $\hat{M}(\omega)$.
4) We apply inverse Fast Fourier transform to the product $\hat{M}(\omega)$ (computed on the previous step). As a result, we get a function $M(x)$.
5) Finally, we reconstruct $\mu(t)$ as $(M(t))^{1/p}$.

Number of computational steps. Let us estimate the number of computational steps of this algorithm. Stages 1, 3, and 5 require linear time ($O(n)$ steps each, so, $O(n)$ total). Stages 2 and 4 involve FFT and therefore, require the time $O(n \log(n))$. Therefore, the total number of computational steps is equal to $O(n) + O(n \log(n)) = O(n \log(n))$, which is much smaller than the $O(n^2)$ time that is needed for straightforward computations.

Comment. A similar algorithm can be applied for computing the sum of more than two fuzzy numbers. Alternatively, we can first use the above algorithm to add the first two of these fuzzy numbers, then add the third one to the result, etc.

IV. Relation Between Estimating Time Complexity and Zadeh's Extension Principle: Analysis and the Resulting Algorithm

What we want. We know that for the problem of computing expression (8), there is an efficient algorithm which is faster than a straightforward $O(n^2)$ algorithm. We would like to use to use this algorithm to come up with a similar faster algorithm for computing the desired expression (2).

Analysis of the problem. The main difference between the desired formula (2) and the formula (8) that describes Zadeh's extension principle is that:

- the desired formula (2) uses addition, while
- the formula (8) corresponding to Zadeh's extension principle use multiplication.

Another difference is that:

- the desired formula (2) uses minimum, while
- the formula (8) corresponding to Zadeh's extension principle use maximum.

Thus, to reduce our problem to the problem of computing Zadeh's extension principle, we must reduce addition to multiplication, and minimum to maximum.

How to reduce addition to multiplication: reminder. It is well known how to reduce addition to multiplication: use an exponential function $\exp(k \cdot x)$ since

$$\exp(k \cdot (a+b)) = \exp(k \cdot a) \cdot \exp(k \cdot b).$$

We want the resulting value $\exp(k \cdot x)$ to be from the interval $[0,1]$ for all $x > 0$. Thus, we must select $k < 0$ – otherwise, we will get values $\exp(k \cdot x) > 1$. The simplest such value is $k = -1$.

The function $\exp(-x)$ is decreasing, so it automatically reduced minimum to maximum.

Resulting reduction: idea. To compute the value (2), we consider the functions $\mu_a(t_a) = \exp(-\Delta_a(t_a))$, $\mu_b(t_b) = \exp(-\Delta_b(t_b))$, and $\mu(t) = \exp(-\Delta_c(t))$.

By definition (2), $\Delta_c(t)$ is the smallest of possible values $\Delta_a(t_a) + \Delta_b(t - t_a)$ corresponding to all possible t_a. Since the function $\exp(-x)$ is decreasing, its values at the smallest of the arguments is the largest, i.e.,

$$\mu(t) = \exp(-\Delta_c(t)) = \max_{t_a} \exp(-(\Delta_a(t_a) + \Delta_b(t - t_a)).$$

Here,

$$\exp(-(\Delta_a(t_a) + \Delta_b(t - t_a)) =$$
$$\exp(-(\Delta_a(t_a)) \cdot \exp(-\Delta_b(t - t_a)) =$$
$$\mu_a(t_a) \cdot \mu_b(t - t_a),$$

hence

$$\mu(t) = \exp(-\Delta_c(t)) = \max_{t_a} \mu_a(t_a) \cdot \mu_b(t - t_a).$$

This is exactly the formula (8).

Once we know $\mu(t) = \exp(-\Delta_c(t))$, we can reconstruct $\Delta_c(t)$ as $\Delta_c(t) = -\ln(\mu(t))$.

Thus, we arrive at the following algorithm.

New algorithm for computing the expression (2). Once we know the accuracy $\Delta_a(t_a)$ with which we can compute a during time t_a and the accuracy $\Delta_b(t_b)$ with which we can compute b during time t_b, to compute a similar characteristic for $c = a + b$, we do the following:

- form functions $\mu_a(t_a) = \exp(-\Delta_a(t_a))$ and $\mu_b(t_b) = \exp(-\Delta_b(t_b))$;
- apply a fast algorithm for computing the fuzzy expression (8) to these functions $\mu_a(t_a)$ and $\mu_b(t_b)$, and thus compute a new function $\mu(t)$;
- compute $\Delta_c(t) = -\ln(\mu(t))$.

Discussion. The fact that we succeeded in relating computation time restrictions and fuzziness is probably not accidental: as noted in [1], [2], in critical situations time is too short to perform exact computations, a good idea is to rely on (fuzzy) expert intuition.

V. From Addition to the General Case

Formulation of the problem. In the above text, we only considered the simplest case of data processing, when we have only two inputs a and b and we compute $c = a + b$. In the general case, we may have several inputs $a_1, \ldots, a_m$, and we compute a more general expression $c = f(a_1, \ldots, a_m)$.

Analysis of the problem. Once we spend time t_i on computing each quantity a_i, we thus get an approximate value $\widetilde{a}_i$ with accuracy $\Delta_i(t_i)$. In other words, we know that the approximation error $\Delta a_i \stackrel{\text{def}}{=} \widetilde{a}_i - a_i$ is bounded by the accuracy: $|\Delta a_i| \le \Delta_i(t_i)$. Once we apply the algorithm f to the estimates $\widetilde{a}_i$, we get an approximate value $\widetilde{c} = f(\widetilde{a}_1, \ldots, \widetilde{a}_m)$.

What is the accuracy of this approximation, i.e., what is the difference

$$\Delta c = \widetilde{c} - c = f(\widetilde{a}_1, \ldots, \widetilde{a}_m) - f(a_1, \ldots, a_m)$$

between this approximation and the actual (desired) value c? Here, by definition of the approximation errors, we have $a_i = \widetilde{a}_i - \Delta a_i$, so

$$\Delta c = f(\widetilde{a}_1, \ldots, \widetilde{a}_m) - f(\widetilde{a}_1 - \Delta a_1, \ldots, \widetilde{a}_m - \Delta a_m).$$

Since the approximations are reasonably accurate, we can expand this expression in Taylor series in terms of Δa_i and safely ignore terms which are quadratic and of higher order in terms of Δa_i. As a result, we get the following expression:

$$\Delta c = \sum_{i=1}^{m} c_i \cdot \Delta a_i,$$

where

$$c_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial a_i}(\widetilde{a}_1, \ldots, \widetilde{a}_m).$$

Since we know that $|\Delta a_i| \le \Delta_i(t_i)$, we thus conclude that $|\Delta c| \le \Delta$, where

$$\Delta = \sum_{i=1}^{m} |c_i| \cdot \Delta_i(t_i).$$

Resulting expression. Thus, for a given time t, the best accuracy $\Delta(t)$ that we can attain can be determined as

$$\Delta(t) = \min_{t_1, \ldots, t_m:\ t_1 + \ldots + t_m = t} \ \sum_{i=1}^{m} |c_i| \cdot \Delta_i(t_i). \qquad (9)$$

Reduction to fuzzy computations: idea. The above formula can be similarly reduced to computing the fuzzy expression

$$\mu(t) = \max_{t_1, \ldots, t_m:\ t_1 + \ldots + t_m = t} \ \prod_{i=1}^{m} \mu_i(t_i), \qquad (10)$$

is we take $\mu(t) = \exp(-\Delta(t))$ and

$$\mu_i(t_i) = \exp(-|c_i| \cdot \Delta_i(t_i)).$$

Thus, we arrive at the following algorithm.

New algorithm for computing the expression (2). Once we know, for every input $i = 1, \ldots, m$, the accuracy $\Delta_i(t_i)$ with which we can compute a_i during time t_i, to compute a similar characteristic for $c = f(a_1, \ldots, a_n)$, we do the following:

- compute approximate values $\widetilde{a}_1, \ldots, \widetilde{a}_m$;
- compute values $c_i = \dfrac{\partial f}{\partial a_i}(\widetilde{a}_1, \ldots, \widetilde{a}_m)$
- form functions $\mu_i(t_i) = \exp(-|c_i| \cdot \Delta_i(t_i))$;
- apply a fast algorithm for computing the fuzzy expression (10) to these functions $\mu_i(t_i)$, and thus compute a new function $\mu(t)$;
- compute $\Delta_c(t) = -\ln(\mu(t))$.

Comment. For addition, we use membership functions $\mu_i(t_i) = \exp(-\Delta_i(t_i))$; in the more general case, we use more complex membership functions $\mu_i'(t_i) = \exp(-|c_i| \cdot \Delta_i(t_i))$. These new functions can be described in terms of the addition-related functions $\mu_i(t_i)$ as $\mu_i'(t_i) = (\mu_i(t_i))^{|c_i|}$. It is worth mentioning that an operation that transforms a degree μ into a degree $\mu' = \mu^a$ is well known in fuzzy techniques: it is one of the main methods of dealing with *hedges*. For example [3], [7]:

- "very" is usually interpreted as a transformation $\mu \to \mu^2$ corresponding to $a = 2$, while
- "somewhat" is usually interpreted as a transformation $\mu \to \sqrt{\mu}$ corresponding to $a = 1/2$.

ACKNOWLEDGMENT

The authors are thankful to the anonymous referees for their valuable suggestions.

REFERENCES

[1] T. Brown, *Critical Care: A New Nurse Faces Death, Life and Everything in Between*, HarperStudio, New York, 2010.

[2] T. Brown, "Learning to Talk the Talk in a Hospital", *New York Times*, September 7, 2010, Science Times section.

[3] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 1995.

[4] A. N. Kolmogorov and S. V. Fomin, *Introductory Real Analysis*, Dover, N.Y., 1975.

[5] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT), *Proceedings of the 1996 IEEE International Conference on Fuzzy Systems*, New Orleans, September 8–11, 1996, Vol. 3, pp. 1958–1964.

[6] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, "Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT)", *Fuzzy Sets and Systems*, 1997, Vol. 91, No. 2, pp. 269–277.

[7] H. T. Nguyen and E. A. Walker, *First Course on Fuzzy Logic*, CRC Press, Boca Raton, Florida, 2006.

[8] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 2009.

[9] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.